



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.01 Информатика и вычислительная техника

О Т Ч Е Т

по лабораторной работе № 5

Название: Основы асинхронного программирования на Golang

Дисциплина: Языки интернет программирования

Студент

ИУ6-31Б
(Группа)

(Подпись, дата)

А.В. Палий
(И.О. Фамилия)

Преподаватель

(Подпись, дата)

В.Д. Шульман
(И.О. Фамилия)

Москва, 2024

Цель работы

Изучение основ асинхронного программирования с использованием языка Golang.

Ход работы

Для начала ознакомились с данными в курсе <https://stepik.org/course/54403/info> в разделе "3. Map, файлы, интерфейсы, многопоточность и многое другое".

Задание 1

Вам необходимо написать функцию `calculator` следующего вида:

```
func calculator(firstChan <-chan int, secondChan <-chan int, stopChan <-chan struct{ }) <-chan int
```

Функция получает в качестве аргументов 3 канала, и возвращает канал типа `<-chan int`.

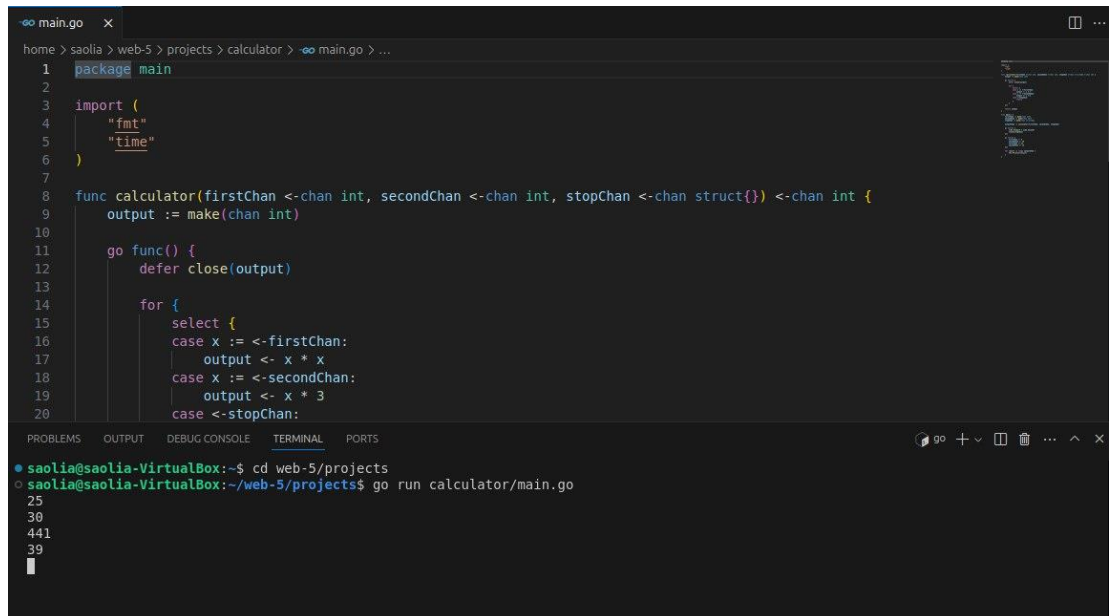
1)в случае, если аргумент будет получен из канала `firstChan`, в выходной (возвращенный) канал вы должны отправить квадрат аргумента.

2)в случае, если аргумент будет получен из канала `secondChan`, в выходной (возвращенный) канал вы должны отправить результат умножения аргумента на 3.

3)в случае, если аргумент будет получен из канала `stopChan`, нужно просто завершить работу функции.

Функция `calculator` должна быть неблокирующей, сразу возвращая управление. Ваша функция получит всего одно значение в один из каналов - получили значение, обработали его, завершили работу.

Результат работы представлен на рисунке 1.



```
1 package main
2
3 import (
4     "fmt"
5     "time"
6 )
7
8 func calculator(firstChan <-chan int, secondChan <-chan int, stopChan <-chan struct{}) <-chan int {
9     output := make(chan int)
10
11     go func() {
12         defer close(output)
13
14         for {
15             select {
16                 case x := <-firstChan:
17                     output <- x * x
18                 case x := <-secondChan:
19                     output <- x * 3
20                 case <-stopChan:
21                     return
22             }
23         }
24     }()
25
26     return output
27 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
saolia@saolia-VirtualBox:~$ cd web-5/projects
saolia@saolia-VirtualBox:~/web-5/projects$ go run calculator/main.go
25
30
441
39
```

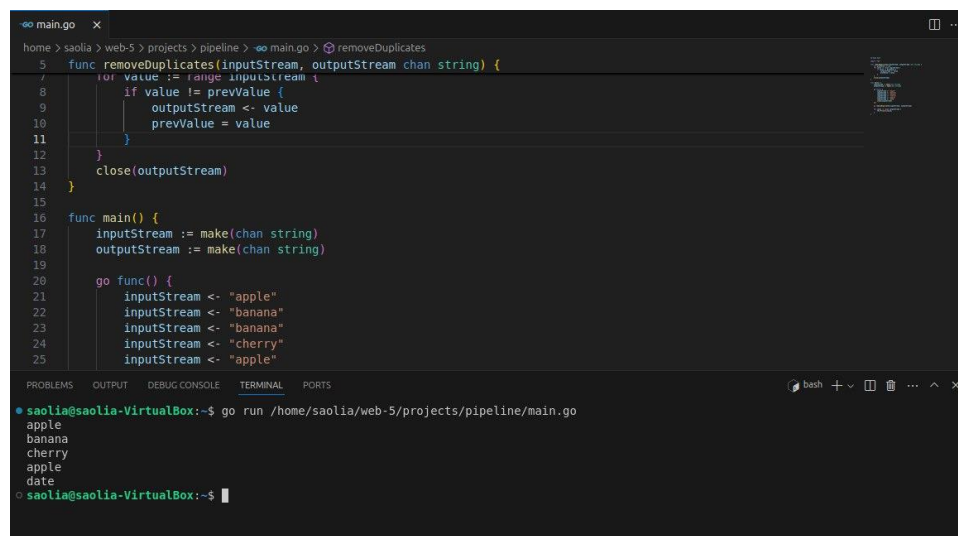
Рисунок 1 - Результат

Задание 2

Напишите элемент конвейера (функцию), что запоминает предыдущее значение и отправляет значения на следующий этап конвейера только если оно отличается от того, что пришло ранее.

Ваша функция должна принимать два канала - `inputStream` и `outputStream`, в первый вы будете получать строки, во второй вы должны отправлять значения без повторов. В итоге в `outputStream` должны остаться значения, которые не повторяются подряд.

Функция должна называться `removeDuplicates()` Результат работы представлен на рисунке 2.



```
5 func removeDuplicates(inputStream, outputStream chan string) {
6     prevValue := ""
7     for value := range inputStream {
8         if value != prevValue {
9             outputStream <- value
10            prevValue = value
11        }
12    }
13    close(outputStream)
14 }
15
16 func main() {
17     inputStream := make(chan string)
18     outputStream := make(chan string)
19
20     go func() {
21         inputStream <- "apple"
22         inputStream <- "banana"
23         inputStream <- "banana"
24         inputStream <- "cherry"
25         inputStream <- "apple"
26     }()
27
28     removeDuplicates(inputStream, outputStream)
29
30     for value := range outputStream {
31         fmt.Println(value)
32     }
33 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

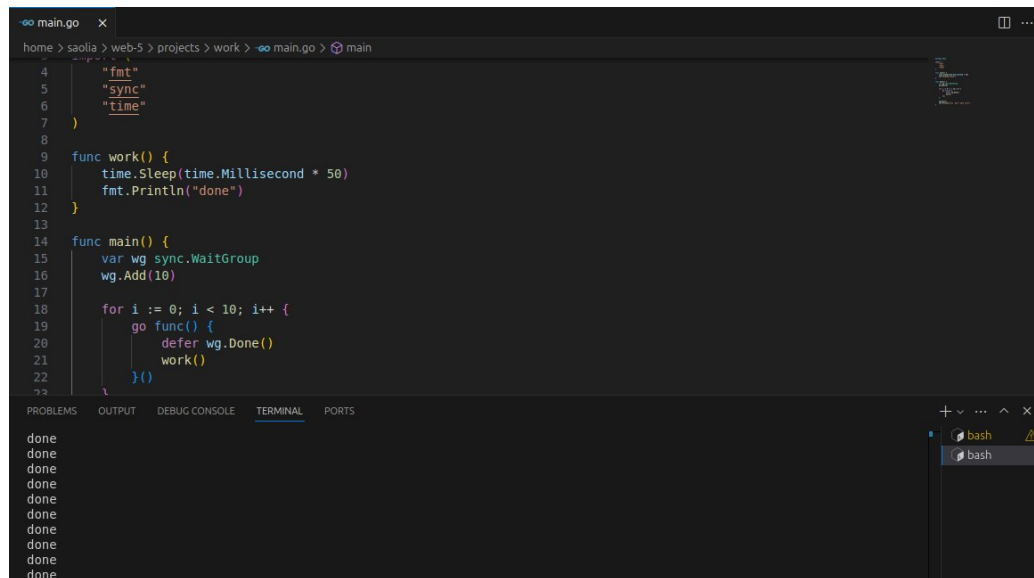
```
saolia@saolia-VirtualBox:~$ go run /home/saolia/web-5/projects/pipeline/main.go
apple
banana
cherry
apple
date
```

Рисунок 2 – Результат

Задание 3

Внутри функции `main` (функцию объявлять не нужно), вам необходимо в отдельных горутинах вызвать функцию `work()` 10 раз и дождаться результатов выполнения вызванных функций.

Результат работы представлен на рисунке 3.

The image is a screenshot of a Go IDE, likely Visual Studio Code, with a dark theme. The editor window shows a Go file named `main.go` with the following code:

```
4  "fmt"
5  "sync"
6  "time"
7  )
8
9  func work() {
10     time.Sleep(time.Millisecond * 50)
11     fmt.Println("done")
12 }
13
14 func main() {
15     var wg sync.WaitGroup
16     wg.Add(10)
17
18     for i := 0; i < 10; i++ {
19         go func() {
20             defer wg.Done()
21             work()
22         }()
23     }
24 }
```

The bottom panel of the IDE is split into two tabs: 'PROBLEMS' and 'TERMINAL'. The 'TERMINAL' tab is active and shows the output of the program, which consists of ten 'done' messages printed on separate lines. The 'PROBLEMS' tab is empty.

Рисунок 3 – Результат

Заключение

В ходе выполнения лабораторной работы я изучила основы асинхронного программирования с использованием языка Golang. Для закрепления знаний было написано три кода.